Page 4 (2nd Paragraph Amended) Figs. 3B-1 and 3B-2 are functional block diagrams of the browser in the second embodiment, where a compressed image file of interlaced format is decompressed, thereby progressively producing images received from the second computer.

Page 4 (Amended 6th Paragraph) Figs. 5B-1 and 5B-2 are network sequence diagrams for the network shown in Fig. 4B.

Page 4 (11th Paragraph Amended) Figs. 11A-1 and 11A-2 are functional block diagrams of the server 110 corresponding to the server process shown in the flow diagram of Fig. 10.

Page 9 (3rd Paragraph Amended) The first user computer **10** includes the control program **20** and second computer **10'** includes control program **20'**, both of which are shown in the network sequence diagram of Fig. 2B-1. The control program **20** in Fig. 2B-1 begins in step **202** with the exchange of hand shake signals including a local public key of the first user computer **10**, which is transmitted to the second computer **10'**. The first computer **10** includes a public key/private key program **40** which is used to generate the enable message **28** to be sent to the second computer **10'**. The public key/private key program **40** uses the first user's private key to sign the enable message **28** uniquely identifying the enable message **28** as being received from the first user computer **10**. The second user computer **10'** includes a public key/private key program **40'** which uses the first users public key to verify that the enable message **28'** was properly signed by the first user's private key. The enable message **28** is then used at the second user computer **10'** to pass from step **202'** to **204'**. Correspondingly, the enable message **28'** sent

from the second user computer **10'** to the first computer **10** is signed by the public key/private key program **40'** in the second computer **10'** using the private key of the second user. The enable message **28'** received at the first user computer **10'** is verified by the public key/private key program **40** in the first computer **10** applying the public key of the second user to verify that the second user signed the enabled message **28'** with the second user's private key. The enable message **28'** then enables the control program **20** in the first computer **10** to flow from step **202** to step **204**. Public key cryptography is described, for example, in the book by Richard E. Smith entitled, "Internet Cryptography", published by Addison-Wesley, 1997.

Page 10 (2nd Paragraph Amended) In step **204** of Fig. 2B-1, the first computer **10** sends the compressed image **F** in message **24** which, in this example, is an interlaced GIF file, to the second computer **10'** for display in step **210'**. Correspondingly, the second computer **10'** sends the compressed image **S** in message **24'**, which in this example, is an interlaced GIF file, to the first computer **10** for display in step **210**. In the control program **20**, step **204** then flows to step **206** which detects the local and remote mouse down states and in the control program **20'**, step **206'** detects the local and remote mouse down states, then step **206** passes a mouse down message **26** to the second computer **10'**. As long as both mouse buttons are pressed on both respective computers **10** and **10'**, step **206** flows to step **208** and step **206'** flows to step **208'**. In step **208**, the first stage enable message signed with a local private key, as previously explained, is sent from the first computer **10** to the second computer **10'**. Correspondingly, the step **208'** in the second computer **10'** sends the first stage enable message **28'** signed with a local private key to the first computer **10**. As shown in Figs. 2B-1 and 2B-2 step **208** flows to step **210** in the first

computer **10**, wherein the interlaced GIF image **S** is decoded in a first pass as the stage 1 picture **S1** shown in Fig. 1B. Correspondingly, in second computer **10'**, the interlaced GIF message **F** is decoded in a first pass as the stage 1 picture **F1**, which the browser **30'** displays as shown in Fig. 1B.

Page 11 (2nd Paragraph Amended) Reference can now be made to Figs. 3B-1and 3B-2 which illustrate a GIF compressed image file S in the interlaced format being progressively decompressed in consecutive stages which can be locally controlled. Fig. 3B-1 shows the first user computer **10** in which the browser **30** in the computer **10** has received and is to display the interlaced GIF file S having the file structure **300**. The interlaced GIF file structure **300**, in this example, is a GIF 89A type file structure which is described in greater detail in the book by James D. Murray, et al., entitled "Graphics File Formats", 2nd Edition, published by O'Reilly and Associates, 1996. The interlaced GIF file structure **300** includes the GIF 89A header **302** which is a small 6 byte character block containing the GIF version of the file. Also included in the file structure **300** is the local screen descriptor **304** which defines an area of pixels for the GIF image on the user's display. The control block **306** of the file structure **300** provides for user input options. Normally, when an interlaced GIF file is received by the browser **30**, the GIF file is immediately rendered on the screen. However, if a user-input signal is required, as specified in the control block **306**, the image rendering must wait until the next stage enable message **28'** is received from the second user computer **10'**. The control program **20** establishes what is to be considered as a user input signal. The image block **308** of the file structure **300** includes the requirement that the image must be interlaced. Interlacing is a way of saving and displaying the

image data. For interlacing to occur, the image must be initially saved in the interlaced format when the image is created. Interlacing saves alternate rows, producing a venetian blind or block-focusing effect, depending upon the browser's handling of interlacing. Interlacing stores the rows of the image in the order as follows:

Page 12 (2nd Paragraph Amended) Reference is made to Fig. 3B-2 which shows stage 1 for picture **S1**, stage 2 for picture **S2**, stage 3 for picture **S3**, and stage 4 for picture **S4**. These stages correspond to the progressively more detailed display of the compressed image S with each pass of the browser **30** through the loop provided from the image block **308** to the control block **306** of the GIF file structure **300**. The flow from the control block **306** to the image block **308** can only be accomplished when a next stage enable message **28'** is received from the second user computer **10'**. The GIF file structure of **300** of Fig. 3B-1 concludes with a trailer **310** as shown in Fig. 3B-2. Other progressive or interlaced image compression file formats can be used as discussed above including JPEG, JBIG, and PNG, all of which are discussed in the above-cited James D. Murray, et al., book.

Page 12 (3rd Paragraph Amended) Referring back to Fig. 2B-2, the step **212** of the control program **20** periodically detects the local and remote mouse down states for the first program **20**, and correspondingly step **212'** periodically detects local and remote mouse down states for the program **20'**. If the first and second users continue to be interested in viewing progressively more detailed images which are decompressed from the compressed images **F** and S, then they continue to hold down their respective mouse buttons (or keyboard key depression,

as appropriate). Step **212** then flows to step **214** and if both mouse down states exist, then control program **20** sends the next stage enable message to the computer **10'**. As detailed above, enable message **28** is signed with the private key of the first user at computer **10** and is sent to the second user computer **10'** to enable the browser **30'** and the second computer **10'** to produce the stage 2 picture **F2**. Step **214** of the control program **20'** correspondingly determines that if both mouse buttons are down, the next stage enable message **28'** is sent from the second user computer **10'** to the first user computer **10**. Enable message **28'** is signed by the private key of the user at the second computer **10'**. When the enable message **28'** is received at the first computer **10** as previously discussed, the next stage picture **S2** will be decoded from the compressed file structure **300** in Fig. 3B-1 and displayed to the first user in computer **10**. Step **216** in the control program **20** decodes and displays the next stage decompressed image from compressed GIF image **S** from the remote computer. Correspondingly, step **216'** in the control program **20'** decodes and displays the next stage decompressed image from compressed GIF image **F** from the remote computer **10**. Step **216** loops back to step **212** in the control program **20** to consecutively decode and display the pictures **S2**, **S3**, and **S4** from the compressed image **S** as long as both mouse buttons are down. Similarly, step **216'** loops back to step **212'** in control program **20'** to progressively decode and display the next pass interlace GIF image for the picture **F2**, **F3**, and **F4** for the progressively decoded GIF file structure for the compressed image **F** received in the second computer **10'**.

Page 13 (2nd Paragraph Amended) Fig. 4B is a network diagram illustrating that the peer to peer relationship between the first computer **10** and the second computer **10'** is established by

the mailbox/chatroom application **1146** in the server **110**, which functions as a chat room mailbox. Figs. 5B-1 and 5B-2 are network session diagrams illustrating the messages shown in Figs. 2B-1 and 2B-2, which pass through the server **110** in the network diagram of Fig. 4B.

Page 13 (Last Paragraph Amended) The browser program **30** in computer **10** and the browser program **30'** in computer **10'** can be, for example, a Microsoft Internet Explorer 5 browser, which is capable of being programmed to perform functions such as described for the handling of the GIF 89A file structure **300** in Fig. 3B-1. Reference is made to the book by Scott Roberts, entitled "Programming Microsoft Internet Explorer 5," Microsoft Press, 1999, for extensive discussion of how to customize the functions of the browsers **30** and **30'** to conform with the functions described in connection with Figs. 3B-1 and 3B-2. Other browsers can be used as browser **30** and/or **30'**, for example, the Netscape Navigator browser, and many other browsers which are available, for example, Hot Java by Sun Microsystems, and Web Explorer by IBM Corporation.

Page 18 (2nd Paragraph Amended) Table 2 shows an example DTD **"resume.dtd"** that is in the **"SYSTEM"** directory of the employer's second computer **10'** as shown in the following Table 2. The example XML Document For An Employment Resume of Table 1 is sent by the job applicant at the first computer **10** to the employer at the second computer **10'**, where the Example DTD **"resume.dtd"** in Table 2 validates each of the document elements of the XML document in Table 1 and groups them into the progressive stages that are passed to the control program **20C'** in the second computer **10'**. Following the job applicant's computer **10**

handshake step **222** of Figure 2C-1, step **224** sends the XML document **23** and the DTD for the resume to the employer's computer **10'**. Step **226** detects the mouse down states. The employer's computer performs similar steps **222'**, **224'**, and **226'**, sending the XML document **23'** and the DTD for the job description.

Page 19 (2nd Paragraph Amended) The prospective employer's computer **10'** will receive the XML document from the job applicant's first computer **10**, and will pass it to the XML processor / parser **27'**. The XML processor / parser **27'** interprets the XML tags and elements in the XML document and organizes the tagged parts into progressive stages for presentation to the user, in accordance with the definition of those stages in the DTD identified for that document. The group stages are passed to the control program **20C'** in the second computer **10'** that progressively presents each stage to the prospective employer, using the process described in the flow diagram of Figures 2C-1 and 2C-2.

Page 19 (3rd Paragraph Amended) Figures 2C-1 and 2C-2 show the job applicant's computer **10** sending an enable message in step **228**, displaying the first pass decoded text for stage 1 in step **230**, periodically detecting mouse down states in step **232** and sending the next stage enable message in step **234**. Step **236** loops to repeatedly decode and display the next stages. The employer's computer **10'** performs similar steps at **232'**, **234'**, and **236'**.

Page 20 (2nd to last Paragraph Amended) Note that the order of occurrence of the elements in the XML document is not important, since the DTD will sort the elements by

progressive stages, as they are defined in the DTD. Also note that there are five progressive stages defined by the DTD. There can be as many stages as needed to convey the desired information, since the control program **20C'** shown in the flow diagram of Figures 2C-1 and 2C-2 keep looping to the next stage until there are no more stages to present, or until one of the mouse-down signals turns off.

Page 23 (2nd Paragraph Amended) The job applicant's first computer **10** will receive the XML document from the employer's second computer **10'**, and will pass it to the XML processor / parser **27**. The XML processor / parser **27** interprets the XML tags and elements in the XML document and organizes the tagged parts into progressive stages for presentation to the job applicant, in accordance with the definition of those stages in the DTD identified for that document. The grouped stages are passed to the control program **20C** that progressively presents each stage to the job applicant, using the process described in the flow diagrams of Figures 2C-1 and 2C-2.

Page 24 (2nd to last Paragraph Amended) Note that the order of occurrence of the elements in the XML document is not important, since the DTD will sort the elements by progressive stages, as they are defined in the DTD. Also note that there are five progressive stages defined by the DTD. There can be as many stages as needed to convey the desired information, since the control program **20C** shown in the flow diagrams of Figures 2C-1 and 2C-2 keep looping to the next stage until there are no more stages to present, or until the mouse-down signal turns off. However, the number of stages in the job applicant's resume document of

Table 1 should be the same as number of stages in the prospective employer's job description document of Table 3, in order that each party will receive progressive information at each stage of the transaction.

Page 29 (Last Paragraph Amended) Figures 11A-1 and 11A-2 are functional block diagrams of the server **110**, showing the memory **1102** of the server **110** storing components of software program objects needed to perform the operations of handling customized discounts and payment plans and handling digital coupons. The memory **1102** of the server **110** is connected by the system bus **1104** to the central processor **1110** that executes the programmed instructions stored in the memory **1102**. Bus **1104** is also connected to the merchant's database **800** and **900**. The TCP/IP network adapter **1106** is connected by the bus **1104** to the memory **1102**, for connecting the server **110** to the first and second computers **10** and **10'**. The banking network adapter **1112** is connected by the bus **1104** to the memory **1102**, for connecting the server **110** to a private banking network and banking servers which can be used by the server **110** to check credit histories and to arrange for credit card, debit card, or E-Cash payments by the user. The central processor **1110** can be, for example, an IBM RS/6000 Enterprise Server, an IBM AS/400e Server, or the like.

Page 30 (Last Paragraph Amended) Figures 11A-1 and 11A-2 show the various functional modules of the server **110** arranged in an object model. The object model groups the various object-oriented software programs into components which perform the major functions and applications in the server **110**. Enterprise Java Beans (EJB) is a software component

architecture for servers, which is suitable for embodying the object-oriented software program components of Figures 11A-1 and 11A-2. A description of E-Commerce server programming applications developed with Enterprise Java Beans is provided in the book by Ed Roman entitled "Mastering Enterprise Java Beans", published by John Wiley and Sons, 1999. A description of the use of an object model in the design of a web server for E-Commerce applications is provided in the book by Matthew Reynolds entitled "Beginning E-Commerce", Wrox Press Inc, 2000, (ISBN: 1861003986). The components of object-oriented software programs in the object model of memory **1102** are organized into a business logic tier **1114**, a presentation tier **1115**, and an infrastructure objects partition **1122**. The business logic tier **1114** is further divided into two partitions: an application services objects partition **1124** and a data objects partition **1126**. The infrastructure objects partition **1122** includes an object-oriented software program component for the database server interface **1130**, an object-oriented software program component for the system administrator interface **1132**, and the operating system **1125**. The operating system **1125** can be, for example, IBM AIX, IBM OS/400, IBM OS/390, Microsoft Windows NT, Red Hat Linux, or Caldera Linux.

Page 31 (First Paragraph Amended) Figures 11A-1 and 11A-2 show the presentation tier **1115** including a TCP/IP interface **1120** and a bank interface **1125**. The presentation tier **1115** manages the graphical user interface with the user. A suitable implementation for the presentation tier **1115** is with Java servlets to interact with the user using the hypertext transfer protocol (HTTP). The Java servlets run within a request/response server, handling request messages from the user and returning response messages to the user. The Java servlet is a Java

object that takes a request as input, parses its data, performs some logic, and then issues a response back to the user. The Java servlets are pooled and reused to service many user requests. The TCP/IP interface **1120**, implemented with Java servlets, functions as a web server that communicates with the users using the HTTP protocol. The TCP/IP interface **1120** accepts HTTP requests from the user and passes the information in the request to the visit object **1128** in the business logic tier **1114**. Result information returned from the business logic tier **1114** is passed by the visit object **1128** to the TCP/IP interface **1120**, which sends the results back to the user in an HTTP response. The TCP/IP interface **1120** exchanges data through the TCI/IP network adapter **1106** of server **110** with the first and second users' computers **10** and **10'**. Java servlets and the development of web site servers is described in the book by Duane K. Fields, et al. entitled "Web Development with Java Server Pages", published by Manning Publications Co., 2000.

Page 31 (Last Paragraph Amended) The business logic tier **1114** in Figure 11A-1 includes multiple instances of the visit object **1128**, **1128'**, and **1128''**. Each user's computer **10** and **10'** that sends a message to the server **110** has a temporary and separate visit object **1128** instantiated to represent the visit. The Enterprise Java Bean server can instantiate multiple copies of the visit object component **1128** in the business logic tier **1114** to handle multiple messages from multiple users. Each visit object **1128** will buffer user-specific information and maintain a user-specific state for the duration of the session with the user. Each visit object **1128** is a "stateful session bean" that will hold the conversational state about the user's visit. A stateful session bean is an Enterprise Java Bean that services business processes that span

multiple method requests or transactions. The stateful session bean retains state on behalf of an individual user. Data received by the server from the user and data sent by the server to the user will be temporarily buffered in the visit object **1128**. Each visit object **1128** receives from the interface **1120** the user data sent by the user's computer **10** or **10'** to the server **110** in step **1101** of Figure 11B. Each visit object **1128** will also buffer the resulting information that is computed by the server and is passed back to the TCP/IP interface **1120**.

Page 32 (2nd Paragraph Amended) Figure 11B shows the allocation of a given request by the visit object **1128** to the various application programs **1140** to **1148** in Figures 11A-1 and 11A-2, depending on the nature of the request to the server. The visit object method **1128** receives the user's request in step **1101** and determines in step **1103** whether so an interest-matching request. If so, then step **1105** sends a method call to the interest matching application **1140** in Figure 11A-1. If not, then the method flows to step **1107**. The visit object method **1128** determines in step **1107** whether the server has received a server control request. If so, then step **1109** sends a method call to the control program application **1142** in Figure 11A-1. If not, then the method flows to step **1111**. The visit object method **1128** determines in step **1111** whether the server has received an auction request. If so, then step **1113** sends a method call to the auction application **1144** in Figure 11A-2. If not, then the method flows to step **1117**. The visit object method **1128** determines in step **1117** whether the server has received a peer-to-peer mailbox request. If so, then step **1119** sends a method call to the mailbox/chatroom application **1146** in Figure 11A-2. If not, then the method flows to step **1121**. The visit object method **1128** determines in step **1121** whether the server has received a payment request. If so, then step **1121**

sends a method call to the payment application **1148** in Figure 11A-2. If not, then the method flows to step **1127**, which sends the request to a parser for additional processing.